DLQ step 1- Documentation

# Overview

This Python script simulates methane gas emissions using data from multiple sensors and meteorological inputs. It processes raw sensor data, handles wind information, and runs simulations to estimate methane concentrations at various receptor locations. The calculations consider factors like wind speed, wind direction, and distance between emission sources and receptors. The code is optimized for parallel execution across multiple CPU cores, making it ideal for large datasets.

## Features

- Data cleaning and interpolation for missing sensor values.  
- Simulation of methane emissions using a multi-source model.  
- Utilizes wind speed and direction data for accurate concentration predictions.  
- Parallel processing for efficient computation.  
- Flexible output formats (CSV, Pickle).

# Requirements

Before using the script, ensure you have the following dependencies installed:  
- pandas: For data manipulation.  
- numpy: For numerical operations.  
- scipy: For interpolation and scientific computations.  
- joblib: For parallel processing.  
- tqdm: For progress bars.  
- pytz: For timezone management.  
- multiprocessing (Python standard library): For parallel execution.

You can install the required libraries using the following command:

pip install pandas numpy scipy joblib tqdm pytz

# Usage Instructions

1. \*\*Input Data\*\*: Place the CSV files in the same directory where the script is executed or update the file paths in the script accordingly. The required files include:  
 - \*\*Sensor Data\*\*: CSV with methane concentrations and timestamps.  
 - \*\*Source Locations\*\*: CSV with latitude and longitude of emission sources.  
 - \*\*Receptor Locations\*\*: CSV with coordinates where methane concentrations are to be measured.

2. \*\*Execution\*\*: Run the script with:

python simulate.py

Ensure the file paths are correctly specified in the script if the input files are in different locations.

# Input and Output Examples

## Example Input:

- \*\*Sensor Data CSV\*\*: Timestamps and methane concentrations from different sensor locations. (ADED\_Clean.CSV)  
- \*\*Source Locations CSV\*\*: Latitude and longitude coordinates of emission sources.  
- \*\*Receptor Locations CSV\*\*: Latitude and longitude of receptor locations where methane concentrations will be computed.

## Example Output:

- \*\*aligned\_data.csv\*\*: Cleaned and time-aligned data from sensors and wind sources.  
- \*\*simulation\_output.pkl\*\*: Pickle file with serialized simulation results.  
- \*\*final\_concentrations.csv\*\*: CSV with predicted methane concentrations at receptor locations.

# Key Functions

## process\_row(t)

Processes a single timestamp row from the dataset, extracting methane concentrations for use in simulations. Handles missing sensor data with NaN values.

### Parameters:

* t: Timestamp to process.

### Returns:

* A dictionary containing methane concentration data for each sensor at the specified timestamp.

## run\_source\_simulations()

Runs methane emission simulations based on sensor and source location data. Computes methane concentrations at receptor locations, taking into account emission rates and wind data.

### Parameters:

- \*\*data\*\*: Cleaned and time-aligned sensor data.  
- \*\*source\_locs\*\*: List of (latitude, longitude) coordinates for emission sources.  
- \*\*sensor\_locs\*\*: Receptor locations for methane concentration measurement.  
- \*\*emission\_rate\*\*: Rate of methane emissions from sources.  
- \*\*dt\*\*: Time step interval for each simulation.  
- \*\*num\_cores\_to\_use\*\*: Number of CPU cores to utilize for parallel processing.  
- \*\*output\_directory\*\*: Directory for saving output files (CSV and pickle).  
- \*\*chunk\_size\*\*: Number of rows to process in each batch.

### Returns:

CSV files containing receptor concentrations and a Pickle file with detailed simulation results.

# Parallel Processing and Optimization

This script is optimized for large-scale datasets through parallel processing, utilizing Python’s multiprocessing and joblib libraries. The data is processed in chunks, allowing for efficient use of CPU resources across multiple cores. The number of cores can be adjusted by modifying the num\_cores\_to\_use parameter in the run\_source\_simulations function.

# Data Processing and Simulation Logic

## Data Cleaning

The script reads sensor and wind data from CSV files, interpolates missing values, and synchronizes the data based on timestamps. This ensures consistency across datasets before simulation.

## Wind Data Handling

Wind speed and direction are critical for the simulation. The script adjusts wind direction into radians for accurate directional calculations.

# Customization

## Simulation Parameters

Key parameters can be adjusted in the script, such as:  
- \*\*num\_cores\_to\_use\*\*: Modify the number of CPU cores for parallel processing.  
- \*\*emission\_rate\*\*: Set the emission rate for methane sources.  
- \*\*chunk\_size\*\*: Adjust the number of rows processed in each batch.

## File Paths

If the input files are not in the default directory, update the file paths in the script at the top of the file.

# Output Files

## aligned\_data.csv

Contains clean and time-aligned sensor, wind speed, and wind direction data.

## simulation\_output.pkl

Serialized (pickled) output containing the simulation results.

## final\_concentrations.csv

CSV file with predicted methane concentrations at each receptor location.

## Issue Faced with float64 and Value Lag in Python Compared to R

**Context:**

During the development of the DLQ code, there was a noticeable difference between the values calculated by the Python implementation and the R code. Specifically, when comparing concentration values, Python values lagged behind the corresponding R values by a small margin.

**Cause:**

After investigation, the issue was traced to the precision of the floating-point numbers used in the calculations. The Python code was using the float64 datatype by default, which led to slight discrepancies in precision when compared to the R code.

**Solution:**

To address this issue, I converted the data to regular float values in Python, which improved the accuracy of the computed values. As a result, the values from Python matched exactly with the R outputs.

To verify this, a comparison between the outputs from R and Python was conducted. Below is the R code used for comparison and a plot that demonstrates the exact match between R and Python.

**Code for Comparison:**

library(tidyverse)

library(here)

## Read R and Python output data

r <- read\_csv(here("code", "simulation\_outputs\_new", "source\_1", "8\_final\_concentrations.csv")) |>

select(-1) |>

mutate(version = "R")

py <- read\_csv(here( "code", "simulation\_outputs\_new", "source\_1", "py.csv"))|>

mutate(version = "Py",

time = time - hours(6)) # Adjust time for Python

## Combine data

data <- bind\_rows(r, py)

## Reshape data for plotting

data2 <- bind\_rows(r, py) |>

pivot\_longer(cols = -c(time, version))

## Plot comparison between R and Python

ggplot(data2, aes(x = time, y = value, col = version)) +

geom\_line() +

facet\_wrap(~ name) +

labs(title = "Comparing R and Python Versions") +

theme\_minimal()

**Visualization:**

The following plot demonstrates the comparison between R and Python versions. The Python values (red line) now match the R values (blue line) exactly after addressing the float64 issue:

A graph with numbers and lines

Description automatically generated A graph of a graph

Description automatically generated

Conclusion

This script is a powerful tool for simulating methane emissions and calculating gas concentrations based on sensor and wind data. With its parallel processing capabilities, it can handle large datasets efficiently, making it suitable for large-scale environmental monitoring projects.